IMPROVING OPPONENT INTELLIGENCE THROUGH OFFLINE EVOLUTIONARY LEARNING

Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma
Universiteit Maastricht IKAT
P.O. Box 616
NL-6200 MD Maastricht, The Netherlands
E-mail: p.spronck@cs.unimaas.nl

KEYWORDS
Gaming, commercial computer games, handheld computers, artificial intelligence, machine learning, evolutionary systems, neural networks.

ABSTRACT
Artificially intelligent opponents in commercial computer games are almost exclusively controlled by manually-designed scripts. With increasing game complexity, the scripts tend to become quite complex too. As a consequence they often contain “holes” that can be exploited by the human player. The research question addressed in this paper reads: How can evolutionary learning techniques be applied to improve the quality of opponent intelligence in commercial computer games? We study the offline application of evolutionary learning to generate neural-network controlled opponents for a complex strategy game called PICOVERSE. The results show that the evolved opponents outperform a manually-scripted opponent. In addition, it is shown that evolved opponents are capable of identifying and exploiting holes in a scripted opponent and exhibiting original tactical behaviour. We conclude that evolutionary learning is an effective tool to improve the quality of opponent intelligence in commercial computer games.

INTRODUCTION
The aim of opponents in commercial computer games is to provide an entertaining playing experience rather than to defeat the human player at all costs (Tozour 2002). The quality of the opponents in games such as computer role-playing games (CRPGs), first-person shooters (FPSs) and strategy games, lies primarily in their ability to exhibit intelligent behaviour. This implies that computer-controlled opponents should at least meet the following four requirements: they should (1) not cheat, (2) exploit the possibilities offered by the environment, (3) learn from mistakes, and (4) avoid clearly ineffective behaviour. Opponents in today’s computer games, however, have not yet reached this level of behaviour. The appeal of massive online multi-player games stems partly from the fact that computer-controlled opponents often exhibit what has been called “artificial stupidity” (Schaeffer 2001) rather than artificial intelligence.

In early CRPGs and most of present-day FPSs and strategy games, an opponent’s behaviour is usually determined by a straightforward script such as “attack the target if it is in range, else move towards the target in a straight line.” However, more advanced games contain opponents controlled by large scripts comprising hundreds of complex rules (Brockington and Darrah 2002). As any programmer knows, complex programs are likely to contain bugs and unanticipated features. As a consequence, intelligent opponents intended to pose a considerable challenge to a human player often suffer from shortcomings that are easily recognised and exploited. For example, in the CRPG SHADOWS OF AMN (2000; illustrated in figure 1) the dragons, the supposedly toughest opponents in the game, could be easily defeated by taking advantage of holes in the extensive scripts controlling their actions. Evidently, such artificial stupidity spoils the playing experience.

State-of-the-art artificially intelligent opponents lack the ability to learn from experience. Therefore, the research question addressed in this paper reads: How can evolutionary learning techniques be applied to improve the quality of opponent intelligence in commercial computer games? We discuss two main ways of applying machine learning to games: offline learning and online learning. We introduce the strategy game PICOVERSE and outline the duelling task for which we evolve opponent intelligence offline. We then describe the environment and techniques we have used for our initial experiments. We present the results of two series of experiments and discuss them. Finally, we draw some conclusions and point out future research.

OPPONENT INTELLIGENCE LEARNING
We distinguish two main ways of applying machine learning to improve the quality of opponent intelligence in commercial computer games: online learning and offline learning.
Online Learning

Examples of online application of machine learning are some of the opponents developed for the popular FPS QUAKE. The artificial player in QUAKE III (commonly called a “bot”) uses machine learning techniques to adapt to its environment and to select short-term and long-term goals (Van Waveren and Rothkrantz 2001). John Laird has developed a bot that predicts player actions and uses these predictions to set ambushes and to avoid traps (Laird 2001). Of the four requirements we mentioned in the introduction for opponent strategies that exhibit high entertainment value, these bots address the first two, namely managing to avoid cheating and using their environment effectively. However, they can not learn from mistakes or generate completely new tactics to overcome ineffective behaviour. They mainly adapt to the world they find themselves in, rather than to the tactics of the human player. Still, these bots are a first step towards the creation of opponents with human-like qualities by online adaptation.

Machine learning is rarely used in commercial computer games (Tozour 2002). Presumably, the widespread dissatisfaction of game developers with machine learning (Woodcock 2000) is caused by the bold aim of creating intelligent opponents using online learning. Machine learning techniques require numerous experiments, generate noisy results, and are computationally intensive. These characteristics make machine learning rather unsuitable for online adaptation of opponents in computer games.

Offline Learning

In the offline application of machine learning techniques the disadvantages mentioned for online learning do not pose an insurmountable problem. However, to our knowledge, developers of commercial games have never used machine learning for offline learning. In our view the two main applications of offline learning in games are: (1) to enhance intelligence of opponents by training them against other (scripted) opponents, and (2) to give opponents some resilience against unforeseen player tactics by detecting “holes” in the scripts controlling the opponents. Because human players are not involved when offline learning takes place, obviously it is impossible to use offline learning to let opponents adapt to completely new human player tactics. The next six sections describe the experiments supporting our view on the offline application of machine learning in games.

DUELLING SPACESHIPS

In our experiments, we apply offline learning to optimise the performance of opponents in a strategy game called PICOVERSE. This section discusses the game and the learning task to be used in our experiments. Figure 2 shows three screenshots of the game.

PICOVERSE is a relatively complex strategy game for the Palm (handheld) computer. Our intentions with the development of this game are twofold: (1) we use it to support and illustrate our views on the design of complex Palm games (Spronck and Van den Herik 2002), and (2) in the present context, we use it to investigate the application of machine learning to improve opponent intelligence.

In PICOVERSE the player assumes the role of an owner of a small spaceship in a huge galaxy. Players act by trading goods between planets, going on missions and seeking upgrades for their spaceship. During travel, players encounter other ships and combat may ensue. The ships are equipped with laser guns to fight opponent ships. They are protected from destruction by their hulls. Modelling ship damage, the strength of the hull decreases when hit by laser beams. A ship is destroyed when its hull strength is reduced to zero. The duels in PICOVERSE are more strategically oriented than action oriented. While the relative attack power and hull strengths of the spaceships are important factors in deciding the outcome of a fight, even overpowered players have a good chance to escape unharmed if their ship is equipped with fast and flexible drives or specific defence measures. To enhance immersiveness of the game, we permit opponents, who have access to the same equipment as the player, to escape from a duel that they are bound to lose, rather than to continue fighting until being destroyed. This feature makes the opponent intelligence non-trivial, despite the relatively low level of complexity of the game (as compared to state-of-the-art PC games).

OFFLINE LEARNING EXPERIMENTS

In our experiments, the performance of a neural-network controlled spaceship is optimised using offline evolutionary learning in a simplified version of PICOVERSE. For both the evolved and opponent ships, lasers fire automatically when their enemy is within a certain range and within a 180-degree arc at the front of the ship. If a ship bumps head-on into the other ship, its speed is reduced to zero. The movement of the ships is turn-based: movements of the evolved and opponent ships are executed in an alternating sequence. The evolved ship is allowed to move first and the opponent ship is always allowed a last move even if its hull strength is reduced to zero. Two reasons made us prefer a turn-based approach over a simultaneous approach to the combat sequences: (1) a turn-based approach is used in most of today’s strategy games; and (2) a turn-based approach is computationally significantly cheaper than a simultaneous approach, which is an important consideration for time-intensive evolutionary learning experiments.

The neural controllers are trained using evolutionary algorithms. The fitness is determined by letting the evolved spaceships combat against scripted opponents in a duelling task. Below, we discuss the duelling task, the neural network controlling the spaceship and the evolutionary algorithm.

The Duelling Task

Figure 3 is an illustration of the duelling task. We refer to the scripted ship as “the opponent” and to the ship that is
The opponent attempts to match its speed to the speed of the evolved ship. While the opponent ship is firing its laser, it follows. The opponent starts by increasing its speed to maximum and rotating the ship’s nose towards the centre of the evolved ship. While the opponent ship is firing its laser, it attempts to match its speed to the speed of the evolved ship. If the ratio of the current and maximum hull strength of the opponent is lower than the corresponding ratio of the evolved ship, the opponent ship attempts to flee by turning around and flying away at maximum speed. This simple yet effective script mimics a basic strategy often used in commercial computer games.

The Neural Controller

The neural network controlling the (to be) evolved ship has ten inputs. Four inputs represent characteristics of the evolved ship: the laser power, the laser range, the hull strength, and the speed. Five inputs represent characteristics of the opponent ship: the location (direction and distance), current hull strength, flying direction, and speed. The tenth input is a random value. The network has two outputs, controlling the acceleration and rotation of the evolved ship. The hidden nodes in the network have a sigmoid activation function. The outputs of the network are scaled to ship-specific maximums.

We studied two types of neural networks, namely feedforward and recurrent networks. The feedforward networks include fully-connected networks (every neuron may be connected to any other neuron, as long as a feedforward flow through the network is guaranteed) and layered networks (neurons are only connected to neurons in the next layer). The recurrent neural networks are layered networks in which recurrent connections are only allowed between nodes within a layer. Recurrent connections function as a memory by propagating activation values from the previous cycle to the target neuron.

The Evolutionary Algorithm

An evolutionary system, implemented in the ELEGANCE simulation environment (Spronck and Kerckhoffs 1997), was used to determine the neural network connection weights and architecture. All simulations are based on the following settings: a population size of 200, an evolution run of 50 generations, real-valued weight encoding, size-2 tournament selection, elitism, Thierens’ method of dealing with competing conventions (Thierens et al. 1993) and size-3 crowding. As genetic operators we used biased weight mutation (Montana and Davis 1989), nodes crossover (Montana and Davis 1989), node existence mutation (Spronck and Kerckhoffs 1997), connectivity mutation (Spronck and Kerckhoffs 1997), and uniform crossover. In addition, we added randomly generated new individuals to prevent premature convergence.

The fitness is defined as the average result on a training set of fifty duels between the evolved ship and its opponent. Each duel lasts fifty time steps. Each duel in which the ships start with different characteristics is followed by a duel in which the characteristics are exchanged between both ships. At time step \( t \) the fitness is defined as:

\[
\text{Fitness} = \begin{cases} 
0 & \text{if } PH_H \leq OH_H \\
\frac{PH_H}{PH_0} + \frac{OH_H}{OH_0} & \text{if } PH_H > 0
\end{cases}
\]

where \( PH_H \) is the hull strength of the evolved ship at time \( t \) and \( OH_H \) is the opponent hull strength at time \( t \). The overall fitness for a duel is determined as the average of the fitness values at each time step.

Determining the fitness in this way has the following properties. If the evolved ship and its opponent both remain passive the fitness is equal to 0.5. If the opponent ship is damaged relatively to a larger extent than the evolved ship, the fitness is larger than 0.5 and if the reverse is true (or when the evolved ship is destroyed) the fitness is smaller than 0.5. Therefore, the fitness function favours attacking if it leads to victory and favours fleeing otherwise.

The Experiments

Two series of experiments were performed. The first series intended to determine the most suitable architecture for the neural controller and to discover possibilities for improvement of the script that drives the opponent. Specifically, we were interested in exploitability “holes” in the script, and in new tactics that allow the evolved ship to defeat the opponent. In the second series of experiments we intended to improve the script by incorporating the lessons learned from the first series of experiments and by determining the effects of these improvements. The results of the two series of experiments are described in the next four sections.

THE FIRST SERIES OF EXPERIMENTS

Table 1 presents the results of the two types of networks evaluated in the first series of experiments. Evidently, two-layered feedforward neural networks outperform all other networks in terms of both average and maximum fitness.
values. The network with five nodes in each hidden layer did not score significantly better than the network with ten nodes in each layer.

At first glance the best fitness results achieved are not very impressive. A fitness of 0.5 means that the neural controller results are as effective as the manually-designed algorithm. A fitness of 0.579 (the best result obtained in the experiments) may be taken to indicate that the evolved opponent scores only slightly better than the scripted opponent. Since the scripted opponent employs a fairly straightforward tactic, one would expect the neural controller to be able to learn a far more successful tactic. However, a controller that always remains passive reaches a fitness of 0.362. Assuming that a scripted opponent performs better than a stationary ship, a fitness of 0.638 can be considered an upper bound to the maximum the neural controller can reach. From that point of view a fitness of 0.579 is not bad at all.

From the perspective of playing experience, the fitness rating as calculated in our experiments is not as important as the objective result of a fight. A fight can end in a victory, a defeat, or a “draw.” For the best controller, we found that 42% of the encounters ended in a victory for the evolved ship, 28% in a defeat, and 30% in a draw. This means that 72% of the encounters ended in a situation not disadvantageous to the evolved ship. The evolved ship achieved 50% more victories than the opponent ship. Clearly, on the training set the evolved ship performs considerably better than the opponent ship.

**IMPROVING THE SCRIPT**

The results of the first series of experiments show that machine learning (i.e., offline learning) can be used to create intelligent opponents that outperform scripted ones. Analysing the behaviour of the best-performing spaceship, we observed that it showed appropriate following behaviour when it overpowered the opponent. In our experiments, such following behaviour can never be detrimental to the performance. The reason for this is that the opponent’s script ensures that it will only turn around to attack again if the hull strength of the attacker becomes less than its own hull strength, which does not happen as long as the evolved ship stays behind the opponent.

As we expected the evolved ship avoided bumping against the opponent while following it. Avoiding bumping is appropriate behaviour because bumping reduces the evolved ship’s speed to zero while leaving the opponent’s speed unaffected, potentially allowing it to escape. However, contrary to our expectation the evolved ship did not avoid bumping by reducing its speed when approaching the opponent, but by swerving as much as needed to keep a constant relative distance to the opponent.

We further noticed that the evolved ship did not try to flee when losing a fight. The probable reason is that for a spaceship to flee, it must turn its back toward the enemy. The fleeing ship then becomes a target that does not have the ability to fight back (since lasers only fire from the front of the ship). As a result, fleeing ships are almost always destroyed before being able to escape. Such attempts to escape seem therefore of little use. From this observation we conclude that a better balance between the power of the weapons and the versatility of the ships is required to enable effective escaping behaviour.

**Improving the Opponent**

A surprising type of behaviour was observed when the opponent ship started behind the evolved ship, as illustrated in figure 4. In such cases, the evolved ship often attempted to

---

**Table 1: Experimental results of the first series of experiments.** From left to right, the columns indicate the type of neural network tested, the number of experiments performed with the neural network, the average fitness, the lowest fitness value and the highest fitness value. The best results are in boldface.

<table>
<thead>
<tr>
<th>Neural network type</th>
<th>Exps</th>
<th>Average</th>
<th>Lowest</th>
<th>Highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrent, 1 layer, 5 hidden nodes</td>
<td>5</td>
<td>0.516</td>
<td>0.459</td>
<td>0.532</td>
</tr>
<tr>
<td>Recurrent, 1 layer, 10 hidden nodes</td>
<td>5</td>
<td>0.523</td>
<td>0.497</td>
<td>0.541</td>
</tr>
<tr>
<td>Recurrent, 2 layers, 5 nodes per layer</td>
<td>7</td>
<td>0.504</td>
<td>0.482</td>
<td>0.531</td>
</tr>
<tr>
<td>Feedforward, 7 hidden nodes</td>
<td>5</td>
<td>0.472</td>
<td>0.382</td>
<td>0.527</td>
</tr>
<tr>
<td>Feedforward, 2 layers, 5 nodes per layer</td>
<td>5</td>
<td><strong>0.541</strong></td>
<td>0.523</td>
<td><strong>0.579</strong></td>
</tr>
<tr>
<td>Feedforward, 2 layers, 10 nodes per layer</td>
<td>8</td>
<td><strong>0.537</strong></td>
<td>0.498</td>
<td><strong>0.576</strong></td>
</tr>
<tr>
<td>Feedforward, 3 layers, 5 nodes per layer</td>
<td>7</td>
<td>0.515</td>
<td>0.446</td>
<td><strong>0.574</strong></td>
</tr>
</tbody>
</table>

---

**Figure 4:** The opponent starts behind the evolved ship.

Figure 5: The right panel displays a trace of the movements of the evolved ship up to the moment that it fires its first shot. The opponent is overpowered and tries to flee, but the learning ship follows, as shown in the left panel. In this case the opponent is not able to escape.
increase the distance between the two ships, up until the point where a further increase in separation would imply a draw. At that point, the evolved ship turned around and either repeated the behaviour or started to attack. Figure 5 illustrates this sequence of events.

An explanation for the success of the observed behaviour is that if the distance between the two ships is maximal, the evolved ship will have a maximal amount of time to turn around and face the opponent before it gets within the opponent’s laser range. Since facing the opponent is required to counter-attack, the observed behaviour is beneficial to the evolved ship’s strategy. Therefore, improving the script of the opponent accordingly could improve its quality.

Detecting Shortcomings in the Script

By using offline learning, we could also detect shortcomings in the scripted opponent. Although we did not specifically design our experiments for this purpose, we found a considerable “hole” in the script controlling the opponent by observing the behaviour of the two duelling ships.

The opponent bases its decision to flee on a comparison between the relative hull strengths (e.g., if the opponent’s ratio between its current and maximum hull strength is lower than the corresponding ratio of the evolved ship, it concludes that it will most likely lose the fight and will attempt to escape). The opponent’s script does not take into account that it is its own turn to act when it makes this decision. If the comparative hull strengths are close to each other, this certainly becomes an important consideration. For instance, if on the initial approach the opponent ship comes within the range of the lasers of the evolved ship before being able to fire its own lasers, it will be damaged while the evolved ship remains undamaged. Regardless of its own power, this would cause the opponent’s initial reaction to be to flee. Since in most cases the opponent would still be able to fire its lasers once, this behaviour had little influence if the opponent significantly overpowered the evolved ship, because it would start to attack again on the next turn. However, if the strengths of the ships were about equal, we found the evolved ship to exploit this weakness of the opponent, by attempting to manoeuvre into a position from which it could fire the first shot. Plugging this hole in the opponent’s script should be a major improvement to its behaviour.

It is noteworthy that in many commercial turn-based games we have observed holes in the opponent AI similar to the hole we discovered in our script. For instance, in many games it is a good tactic for the player to pass game turns until the enemy has approached to a certain distance so that the player can initiate the first attack. Game designers will seldom let computer opponents employ such a tactic because it could lead to a stalemate where both the player and the computer refuse to move, because whoever makes the first move is at a disadvantage. Similarities with trench warfare are striking.

THE SECOND SERIES OF EXPERIMENTS

In the second series of experiments we changed the opponent’s script by incorporating two potential improvements we discovered in the first series of experiments. For the neural controller we decided to use only a feedforward controller with two hidden layers with 10 nodes in each layer. Preliminary experiments revealed the version with 5 nodes in each layer to be not powerful enough to oppose the new opponent. We did not change any aspect of the evolutionary algorithm.

Changes to the Script

We made two changes to the script driving the opponent ship. The first change concerns the decision to flee. Instead of deciding to flee when its ratio between current and maximum hull strength drops below the corresponding ratio of the evolved ship, the opponent assumes it is able to shoot the evolved ship once more before evaluating the ratios. This change effectively removes the possibility for the evolved ship to trick the opponent into attempting to flee because the evolved ship is able to strike first. We call this the “fleeing change”.

The second change concerns the tactic wherein a ship attacked from behind tries to increase the distance between the two ships before turning around. We call this the “aft attack change” which is implemented as follows. If (1) the evolved ship is behind the opponent ship, (2) the opponent ship is undamaged, and (3) its distance to the evolved ship is between 75 and 150 (180 being the distance beyond which the opponent ship would be considered to have escaped), then the opponent ship does not rotate but simply increases its speed to maximum in order to increase the distance between the two ships. If the distance becomes larger than 150, the distance to the evolved ship is considered to be sufficiently large to let the opponent ship turn around safely. If the distance is smaller than 75, the opponent ship is assumed to be unable to outrun the evolved ship, so it turns around anyway. The starting distance between the two ships in all of the 50 training set cases was between 80 and 125. The sequence of events depicted in figure 6 illustrates the operation of the “aft attack change.”
Comparing Scripted Strategies

We investigated the following four different scripted strategies.

- **Strategy 0** is the original script.
- **Strategy 1** is enhanced with the “fleeing change”.
- **Strategy 2** is enhanced with the “aft attack change”.
- **Strategy 3** is enhanced with both the “fleeing change” and the “aft attack change”.

To evaluate the relative strengths of these four strategies, we pitted them against each other. Both the evolved and opponent ships were controlled by one of the four strategies. The results of the cross-comparison are shown in table 2.

<table>
<thead>
<tr>
<th>opponent strategy</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.499</td>
<td>0.481</td>
<td>0.504</td>
<td>0.505</td>
<td>0.497</td>
</tr>
<tr>
<td>1</td>
<td>0.525</td>
<td>0.491</td>
<td>0.500</td>
<td>0.504</td>
<td>0.505</td>
</tr>
<tr>
<td>2</td>
<td>0.501</td>
<td>0.485</td>
<td>0.494</td>
<td>0.489</td>
<td>0.492</td>
</tr>
<tr>
<td>3</td>
<td>0.507</td>
<td>0.487</td>
<td>0.497</td>
<td>0.492</td>
<td>0.496</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.508</td>
<td>0.486</td>
<td>0.499</td>
<td>0.498</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Comparison between the different strategies. The rows represent the strategies employed by the “evolved” ship (note that we call this the “evolved” strategy, but the strategy has not actually evolved – this is simply the strategy used by the ship that moves first, just as the evolved ship did in the learning experiments) and the columns the strategies employed by its opponent. The cells of the table show the resulting fitness of the strategy of the evolved ship, underneath which are shown the number of wins and losses (wins/losses). The right column shows the average fitness over the rows, and the bottom row the average fitness across the columns.

It is clear from table 2 that the four strategies do not greatly differ in strength. This comes as no surprise, because they have very similar implementations. Strategy 1 has the highest average fitness when it controls the evolved ship, and the lowest when it controls the opponent, so this strategy can be considered to be, in some way, the best of the four.

We discuss two unexpected results from table 2. The first unexpected result is that the values on the main diagonal deviate from 0.5, despite the fact that the competing strategies are equal. The deviation is caused by the turn-based handling of the encounters. Since all values on the diagonal are slightly lower than 0.5, one might conclude that in the 50 training set cases the ship to move second has a small advantage over the other ship.

The second unexpected result concerns the the fitness values and the associated win/loss ratios. For instance, “evolved” strategy 0 combined with opponent strategy 2 has a fitness value of 0.504. This value, which is slightly greater than 0.5, indicates that the evolved strategy performs better than the opponent strategy. However, “evolved” strategy 0 has 13 wins while the opponent strategy 2 has 15 wins. Despite the higher fitness value, the opponent strategy appears weaker than the evolved one in terms of number of won games. The explanation is that the fitness is not based on the number of wins and losses, but on the change of the relative hull strengths during a fight. A fast win in a certain situation yields a higher fitness than a slow win in the same situation. As a result, a few fast wins can compensate for a few extra (slow) losses in the fitness rating.

Neural Controller Results

Table 3 shows an overview of the results of the second series of experiments. In these experiments the evolved ship encounters an opponent ship driven by each of the four strategies. The results achieved when using an opponent with strategy 0 were copied from the first series of experiments. Besides evaluating the controllers on the training set of 50 encounters used during the evolution process, we also re-evaluated the best controllers on five test sets containing 50 novel encounters.

Clearly, on the training set the evolved ship outperforms three out of four strategies. Only the opponent driven by strategy 1 (the “fleeing change”) outperforms the evolved ship. Against strategy 1, the evolved ship has an average fitness lower than 0.5, and even in the case of the best result it loses more often than the opponent ship. It is also clear that strategy 2 (the “aft attack change”) does not increase the effectiveness of the opponent ship. This strategy performs even worse than the original (unchanged) strategy 0.

Examining the results of the best controllers on the test sets, we see that the average fitness drops considerably from the original value. This indicates that, unsurprisingly, the evolved ship is optimized too much on the encounters comprising the training set (i.e., it is overfitting the training set). Interestingly, both the fitness and the win/loss ratio drop significantly more for strategy 2 and 3 (the two strategies that contain the “aft attack change”) than for strategy 0 and 1. The neural controller is therefore overfitting more on the training set against strategy 2 and 3 than against strategy 0 and 1. Also, the neural controllers evolved against strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Exps</th>
<th>Average</th>
<th>Lowest</th>
<th>Highest</th>
<th>Win/loss</th>
<th>Avg. on test-set</th>
<th>Avg. win/loss on test-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>0.537</td>
<td>0.498</td>
<td>0.576</td>
<td>19/14</td>
<td>0.490</td>
<td>16/19</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>0.486</td>
<td>0.471</td>
<td>0.528</td>
<td>9/12</td>
<td>0.434</td>
<td>9/20</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0.547</td>
<td>0.479</td>
<td>0.615</td>
<td>16/8</td>
<td>0.476</td>
<td>10/16</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0.517</td>
<td>0.463</td>
<td>0.570</td>
<td>17/11</td>
<td>0.442</td>
<td>13/19</td>
</tr>
</tbody>
</table>

Table 3: Experimental results of the second series of experiments. From left to right, the eight columns represent: (1) the strategy of the opponent ship, (2) the number of experiments performed against this strategy, (3) the average fitness of the evolved ship, (4) the lowest fitness value, (5) the highest fitness value, (6) the number of wins and losses of the evolved controller with the highest fitness value, (7) the average fitness of the best controller re-evaluated on five test-sets, and (8) the average number of wins and losses for the re-evaluation.
0 and 2 (the two strategies that do not implement the “fleeing change”) end up with a significantly higher average fitness on the test sets than the other two strategies. This means that for a neural controller it is easier to deal with a strategy that does not implement the “fleeing change” than with a strategy that does implement it. Therefore, implementation of the “fleeing change” is certain to improve the effectiveness of the opponent script.

DISCUSSION

The second series of experiments shows that implementing the “fleeing change” is a clear improvement to the success of the opponent’s script. On the other hand, implementation of the “aft attack change” seemed to weaken the script somewhat. Does that mean the “aft attack change” is worthless as an opponent strategy? We venture to answer that question in the negative. In computer games, the developer’s goal is not to create the strongest opponent possible, but to create interesting opponents. In a game such as PICOVERSE there should be several different strategies available to opponents, of various strengths and applicable to various situations. The “aft attack change” may increase in effectiveness if a more detailed analysis of the situation in which it works well is available. But even if this analysis is not implemented, it could be worthwhile to have a selection of the opponents implement this strategy anyway, just to be a bit different and with a possibility to offset the player’s expectations.

The Fitness Function

In the second series of experiments we noticed a discrepancy between the fitness results and the ratio of wins and losses. Since in terms of gaming experience the win/loss ratio is a more important measure than the change in hull strength, the fitness function we used should clearly be improved upon for future experiments. The win/loss ratio in itself is not a suitable fitness measure, because it isn’t able to reward small positive changes in the behaviour of the neural controller. However, enhancing the fitness function we used with penalties for losing a fight and extra rewards for winning a fight can ensure a better correspondence between the fitness and the win/loss ratio, while retaining the ability to reward small strategic improvements.

Fixing “Holes” and Generalisation Issues

Implementation of the “fleeing change” amounts to fixing a “hole” in the original script. The “hole” was detected by using an evolutionary algorithm to develop a ship’s controller and examining the behaviour of the most successful evolved controller. Characteristic of “holes” in a script is that the “hole” can be exploited in many different situations, and therefore the evolved controller should be able to generalise exploitation of this “hole” over all potential situations. Since the results of the re-evaluation of the controllers using the test sets were very different from the results on the training set, we must conclude that in our experiments a generalised solution was not found. This is due to the fact that the training set simply is not large enough to cover all relevant situations in the problem domain. We expect that a larger training set would have resulted in a more general controller, but, of course, this would have increased the needed evolution time accordingly. Still, generalisation is not a goal in itself when looking for shortcomings in scripts for game opponents. The experiments can be considered a success as long as the shortcomings that exist are discovered.

Generalisation to Other Games

We have shown how machine learning can be used to improve opponent intelligence in PICOVERSE. Of course, it remains an open question whether our findings generalise to the far more complex commercial PC games. Even the detection of holes in scripted AI, which is obviously much simpler than developing a whole new tactic, may prove to be too difficult if the number of choices at each turn and the number of turns in an encounter are very large. However, we expect for most games that encounters do not last “too long” (to avoid boredom) and the number of choices is not “too large” (to avoid confusion). Even for commercial PC games it should therefore usually be possible to detect AI shortcomings by offline machine learning.

Employing machine learning to design completely new tactics, however, is probably severely limited in its uses. John Laird warns that while neural networks and evolutionary systems may be applied to tune parameters, they are “grossly inadequate when it comes to creating synthetic characters with complex behaviours automatically from scratch” (Laird 2000). On the other hand, Demasi and Cruz (2002) have shown the viability of the fast online evolution of tactically stronger opponents for a simple action game. It can therefore be expected that for a relatively simple game as PICOVERSE machine learning techniques by themselves can be useful in designing strong or surprising tactics. The combination of machine learning with more structured techniques, such as a subsumption architecture (Brooks 1991) or a technique inspired by Laird’s Soar Quakebot (Laird 2001), is likely to lead to more reliable good results within a shorter time, and may therefore also be suitable for more complex environments.

CONCLUSIONS AND FUTURE WORK

By applying offline learning in the computer strategy game PICOVERSE we were able to improve opponent intelligence by detecting shortcomings in the scripted opponent. We conclude that machine learning can be applied offline to improve the quality of opponent intelligence in commercial computer games. We expect the application of offline learning to detect holes in commercial computer game scripts to be feasible.

Our future research will build upon our results with PICOVERSE. The release version of PICOVERSE will be more complex than the simulation we used, and we will run similar experiments on the more complex opponents in that version. For creating new opponent tactics, we intend to explore other machine learning techniques in combination with, for instance, subsumption architectures. In the long run, we hope to apply our techniques to improve opponent intelligence in commercial computer games.
REFERENCES


ELEGANCE is available from http://www.cs.unimaas.nl/p.spronck/.